Visualization on Massively Parallel Computers using CM/AVS

Michael F. Krogh and Charles D. Hansen
Advanced Computing Lab
Los Alamos National Laboratory
Los Alamos, New Mexico 87545
krogh@acl.lanl.gov **hansen@acl.lanl.gov**

Abstract

CM/AVS is a visualization environment for the massively parallel CM-5 from Thinking Machines. It provides a backend to the standard commercially available AVS visualization product. At the Advanced Computing Laboratory at Los Alamos National Laboratory, we have been experimenting and utilizing this software within our visualization environment. This paper describes our experiences with CM/AVS. The conclusions reached are applicable to any implementation of visualization software within a massively parallel computing environment.

1 Introduction

The Advanced Computing Lab (ACL) is one of two DOE High-Performance Computing Research Centers whose mission is to develop a computational environment which brings together high performance computers, networks, and software to focus on Grand Challenge-scale applications for government, academia, and industry. Representative applications include global climate modeling, materials modeling, quantum chromodynamics, Tokamac simulation, and fluid flow through porous media.

The major focus of ACL is to provide an environment for solving such problems. The computational environment currently consists of several supercomputers (CM-5, CM-200, and Cray YMP/2E), experimental computers (Motorola Monsoon and Intel iWarp), and high-end workstations (SGI, Sun, and IBM). These are interconnected with HIPPI, FDDI, and Ethernet networks to other resources such as high speed storage, video equipment, and other networks.

Through the use of supercomputers, such as the massively parallel CM-5 from Thinking Machines Corporation, researchers are able to perform numerical simulations at very high grid resolutions and very fine time granularity. This results in enormous time varying data sets that need to be analyzed. Visualization plays a key role in this analysis.

One of the problems encountered with creating so much data is that it is just as difficult, if not more so, to perform the analysis. Typical workstations alone are not up to the task of performing the visualization in a timely manner, if they can perform it at all. For example, the Global Ocean Model creates on the order 100 MB data sets per time step, and several hundreds to thousand time steps are needed per simulation. This in the future will be coupled to a Global Weather Model that will also create data of similar size. Graphics workstations do not possess the computation speed, memory, disk, or visualization performance to deal affectively with this magnitude of data.

One way to resolve this problem is to bring the supercomputer into the visualization process. The supercomputer has the necessary resources for dealing with the massive quantity of data. It is also where the data is created, thus the data need not move to another machine for analysis. However, massively parallel supercomputers are lacking in visualization software and they do not

have graphics hardware (other than frame buffers). A better approach is to couple the supercomputer to a graphics workstation and use visualization software across both machines. This is the technique used at ACL.

2 Background

Currently, ACL is using the CM-5 and a SGI 4D/380VGX together with CM/AVS as the visualization software. The two machines are interconnected via FDDI and Ethernet at this time and will be interconnected via HIPPI by the summer of 1993. The CM-5 will also be connected to a HIPPI framebuffer at the same time.

We feel that our approach, that of performing distributed visualization across a massively parallel supercomputer in conjunction with a graphics workstation, makes the best use of each machine's capabilities: the CM-5 for data access and filtering, and the SGI for geometry creation and rendering. At the same time, we are exploring geometry creation and rendering on the CM-5 as well.

2.1 CM-5

The Advanced Computing Laboratory (ACL) at Los Alamos National Laboratory has a 1024-node CM-5. The CM-5 is a commercially available massively parallel supercomputer built by Thinking Machines Corporation [1] and consists of 1024 RISC-based processors (SPARC microprocessors) each with 16MB of local RAM. Each processor also has four 64-bit wide vector units which assist in math coprocessing and contain an additional 4MB RAM each for a total of 32GB of main memory for the entire machine. With four vector units up to 128 operations can be performed by a single instruction. This yields a theoretical speed of 128 GFlops for a 1024-node CM-5.

The 1024 node processors can be divided into partitions whose size must be a power of 2. Each partition is controlled by a partition manager (also SPARC microprocessors). The partition managers are responsible for system administration tasks and executing non-parallel code. A user's program is constrained to operating within a partition.

The CM-5 has three internal high-speed networks: the control network, the data network, and the diagnostic network. The control network is used for data operations, such as broadcasts, global operations, and combining operations. It is also used for synchronization and error handling. All selected processors participate in control network operations. The data network is used for routing data between nodes. The diagnostic network is not available to user programs. High-speed I/O devices, such as parallel disk arrays, HIPPI network interfaces, and frame buffers are also attached to the CM-5 data network.

The CM-5 supports data parallel and task parallel programming models[1]. The data parallel programming model performs the same operation on all the selected data elements For example given an array of numbers, a constant could be added to each number. When using the data parallel model, this operation would logically occur simultaneously on each element of the array. Actual hardware may or may not perform this operation simultaneously on all selected data elements. This would depend on whether or not enough physical processors exist for each element in the array. If there are fewer processors than data elements, then multiple elements are assigned to processors.

The execution parallel[2] programming model divides a task up into a number of subtasks that can run concurrently and independently. Some subtasks can occur in parallel while others might

[1] These are sometimes referred to as SIMD and MIMD models.
[2] Sometimes referred to as task parallel

occur serially. For example a set of processors might be used to factor numbers to search for primes. Each processor could be assigned a number to factor asynchronously from the others. When a processor has finished with a given number, it could request another.

Currently, data parallel style programs can be developed using data parallel Fortran (CMF) or data parallel C (C*). Task parallel programs are written in C, C++, and Fortran, and use a message passing library (CMMD) for communications and synchronization.

2.2 AVS

AVS, the Application Visualization System [2], is a widely available commercial visualization environment based on a dataflow model for visualization and process control. The user connects iconic modules, that represent some functionality, together with wires through the modules' input and output ports. The wires represent flow of data between the modules; and the connections and modules together in a particular arrangement is called a network. Besides multiple input and output ports, modules can also have parameters associated with them. Parameters can appear as either graphical widgets, such as sliders, dials, or text entry boxes as well as input ports. AVS, as packaged, ships with over one hundred modules, and others are available by various other sources. Users can also write modules in either C, C++, or Fortran.

Modules are grouped into four categories: data input, filters, data to geometry mappers, and renderers. Data input modules are typically either file readers or data generators, perhaps even entire simulations. Filter modules are used to transform the data from one format to another. Examples include cropping, downsizing, and colorizing. Data can be transformed from numerical format to geometrical format through mapper modules, such as isosurfacers, contourers, or particle tracers. The fourth category, renderers, includes both polygonal renderers as well as image display modules, file creation modules and other output modules.

A feature of AVS is that it supports distributed computing. Modules in a given network can either run on the same computer as the AVS kernel or they can run on a remotely connected computer. Using this model, AVS provides for heterogeneous computing support. Neither the AVS user nor the module writer need worry about varying data formats, communications, or synchronization. The parallel execution model supported by AVS is that modules that are connected in parallel have the potential to run concurrently. AVS does not provide support for parallel computing within the module, although a specific module may use machine/language specific features to achieve such parallelism.

3 CM/AVS

Thinking Machines has ported portions of AVS to the Connection Machine 5. This port, called CM/AVS, includes the necessary software to run remote AVS modules on the CM-5 node processors. It does not include support for the AVS kernel nor its functionality. The user must have a licensed copy of the AVS kernel running on another computer. This could be another workstation or the CM-5 partition manager. What CM/AVS provides is the support mechanisms for running modules remotely on the CM-5 node processors in a data parallel (SIMD) fashion.

CM/AVS does not appear different than normal AVS to the end user. The user must still invoke AVS as usual. Once inside AVS, the user can invoke specially written modules on a CM-5 partition. This appears no differently than invocation of modules on any other remote computer.

CM/AVS only implements the AVS field data type on the CM. As described below, this maps well into the data parallel programming model.

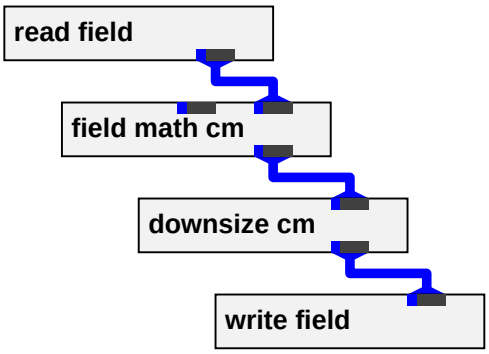

Figure 1: Simple CM/AVS network

A major feature of CM/AVS is that it includes internal support mechanisms for automatically transferring data parallel arrays to serial arrays, and vice versa, when appropriate. This transformation occurs when data is sent into or out of a module. If a data set is sent from a non-CM/AVS module to a CM/AVS module, CM/AVS will automatically put the data set into a parallel array on the node processors. CM/AVS will also transfer the parallel array data back to serial order if the data is to be sent to a non-CM/AVS module. It also will keep the data in parallel order if the data is flowing between two CM/AVS modules.

For example, Figure 1 shows a trivial network that reads in an AVS field on the local workstation (*read field* module), and then sends the data to CM-5 for a simple math operation (*field math cm* module). Next, the data set is sent reduced in size (*downsize cm* module), and then finally sent back to the workstation to be written back out (*write field* module). When the data is sent from *read field* to *field math cm*, it is automatically transformed by CM/AVS into a parallel data structure on the CM-5 nodes. The data set is left in parallel order when flowing between *field math cm* and *downsize cm*, and then is finally converted back to a serial data structure when sent to *write field*.

Rather than utilize the rudimentary task parallel execution model contained within the AVS kernel, CM/AVS employs the data parallel programming model. Modules are not mapped to individual nodes but are spread across as many nodes as are contained in the CM-5 partition where the modules execute. If one were to take the approach of executing each module on a separate node, the majority of the nodes would sit idle for most AVS networks. With the data parallel model, modules are spread across all the nodes within the given partition. More precisely, the *data* is spread across the nodes and is operated upon in parallel by the module.

The module writer can develop CM/AVS modules in either data parallel Fortran (CMF) or data parallel C (C*). Another feature of writing modules for the CM-5 is that the developer need not be aware of the size of the CM-5 partition that the modules will run on. This is a feature of the programming languages.

When developing multiple modules, they can either be either compiled and linked individually into separate executables or they can be linked into a single executable. At this time a single executable containing multiple modules is desirable. This is because modules that are in different executables must communicate data sets through the partition manager. This is can be very time consuming for large data sets. Once node level domain sockets becomes available it will no longer be necessary to send data through the partition manager.

Debugging a CM/AVS module is basically the same process for debugging a normal AVS

module. The user invokes the module using *avs_dbx* and specifies to use either the *prism* debugger or a modified version of *dbx*. Next the developer imports the module into the AVS Network Editor for use. The user can then debug the module as if it were any other CM-5 program.

4 Experiences

At the Advanced Computing Lab we have been using CM/AVS since the product went into beta test. At the time of writing this paper, we have written over 30 modules for CM/AVS, and have gained much insight into CM/AVS, AVS, data parallel visualization, and data flow visualization environments.

CM/AVS has proven to be very reliable. We found very few bugs with the various versions we used. However, we did find several issues that need to be addressed.

4.1 Data Types

AVS has several major data types: primitive types (bytes, integers, floats, and strings), fields (images are a subtype of field), UCD data, colormaps, and geometry[3]. Currently, CM/AVS only supports primitive types, fields, and colormaps[4]. UCD data and, more importantly, geometry are not supported. This has the implication of making it difficult to develop mapper modules and render modules along with all UCD modules.

4.2 Networking

The user of CM/AVS can encounter two serious network bottlenecks if not careful. The first, as previously mentioned, is that two CM/AVS modules will send data directly from one to the other only if they are in the same executable. If they are in different executables, then the data will be sent through the CM-5 partition manager. This can be particularly slow since the partition manager communicates with the nodes through a 400Kb/s link. This problem will be resolved in a later version of CM/AVS which will use parallel domain sockets over the CM-5 data routing network for inter module communication if the two modules if they are not in the same executable.

The other networking bottleneck is when shipping data to or from the CM-5. Currently, we use a SGI graphics workstation connected to the CM-5 via an Ethernet. The size of data sets we wish to analyze are usually tens to hundreds of megabytes. This takes several minutes to transfer across Ethernet. Thus a user needs to be careful when attaching modules executing on the workstation to those executing on the supercomputer. This problem can be magnified when creating geometry on the CM-5. For example, an isosurface of a complex data set could generate much more data than the original data set. In this case it would be better to send the original data set to the workstation for isosurfacing and rendering than it would be to isosurface on the supercomputer and then send the data to workstation for renderering.

One work around is to use a faster network between the two machines. During the summer of 1993 we expect to utilize FDDI and HIPPI networks.

4.3 CM-5 Job Control

In the ACL, access to the CM-5 is controlled through a mechanism known as Distributed Job Manager (DJM). DJM controls resources on MPPs such as the CM-5 and the CM-200. One can request a certain partition size, a specific amount of memory to be used as well as the amount of time needed for the job. Within this environment, it is impossible to perform the **rsh** command. Obviously, this impacts the manner in which CM/AVS modules are invoked. One solution is

to submit an interactive session to DJM. This functionally is the same as a single **rsh** in that it executes a remote shell on the CM-5 partition manager. Once on the partition manager, it is possible to run the AVS Charon daemon and gain access through the mechanism provided. However to the ire of our system managers, this bypasses the accounting and security mechanisms. To remedy this situation, PERL scripts were created which provides a DJM gateway into the CM-5. This script traps the output from the remote module browser and either runs the AVS *listdir* command or submits the selected CM/AVS module through DJM.

As a result of locally enforced DJM limitations, only a single DJM job may be run at any given time. This impacts CM/AVS in two significant ways. First if the user wants multiple CM/AVS modules to execute in the same network, they must be linked into a *single* module. Second, if the AVS kernel has parallel execution turned on, the CM/AVS modules might both be scheduled to execute concurrently which violates the DJM rules and things fail. Currently, we are living with these restrictions and hope to find a more flexible solution in the future.

4.4 Geometry in CM/AVS

As previously mentioned, the AVS geometry data type is not supported in the current CM/AVS implementation. How to work with geometry in a data parallel manner is not a trivial problem. At ACL we experimented with our own geometry type embedded within an AVS field. The hybrid type did not have all of the flexibility of the normal geometry type, but it did allow us to create polygon and polyline lists in parallel. The hybrid type also allowed for optional color and normal information. Once the hybrid geometry-field was sent to the workstation, it was fed into a specialized module that converted the data into a normal AVS geometry type. From there the data could be sent through any of the normal modules that operated on geometry, such as the *geometry viewer*.

The problem we quickly encountered was that we had to send the geometry to the workstation for renderering; and most of the time the geometry took longer to send that it did to send the original data and create the geometry on the workstation. This was especially true for isosurfaces. Particles, however, worked fine due to the small amount of information needed represent particle traces.

4.5 Examples

Figure 2 and Figure 3 demonstrate two ACL applications which utilize the CM/AVS environment. These demonstrate our concept of the best utilization of resources within the distributed visualization environment.

Figure 2 shows a CM-5 molecular dynamics simulation with over 2 million particles. The particles are scattered data, that is, there is no guaranteed order in which the particles appear in the dataset. The *readLomdahl2 cm* reads the data in parallel from the CM-5's DataVault. The *clip cm* module selects which particles to include in the image by clipping to a rectangular window in the data space as well as selecting particles within a given kinetic energy range. The *makeRaster cm* module gathers the scattered particles into rasterized bins. Each of the bins, raster cells, contains the average of all particles which fall into the bin. The output data stream from this module is a 2D floating point field. This is colorized on the CM-5 then sent over the network to the workstation running the AVS kernel. The remaining three modules are run on the local workstation.

Figure 3 is a screen dump from the LANL Global Ocean Model which runs on the CM-5 at the ACL. The image shows the magnitude of the ocean velocity at sea level. All the modules are executing on the CM-5 except for the *generate colormap* module and the *display image* module.

The top two modules read in the U and V components of ocean velocity. Next, the *field math cm* modules are used to calculate the magnitude of the 3D fields. We are demonstrating that the magnitude could be computed without writing a custom module although we could have used a *compute magnitude module cm* which runs on the CM-5 and would have been more efficient. *bottomOut cm* is used with *oceanDepth cm* to place sentinel values into the 3D grid where ever there is not valid ocean data (where land masses, submerged or unsubmerged, reside). The *orthogonal slicer cm* module allows the user to select a 2D slice out of the 3D data set. The remaining modules are used to colorize the slice and display it.

5 Future Work

Several things can be done to improve the utility of CM/AVS. The ACL and Thinking Machines Corporation are jointly exploring solutions to these issues.

5.1 Geometry in CM/AVS

As previously mentioned, the lack of the geometry is limiting since the filtered data sizes are still overwhelming the graphics workstation. Additional work needs to be done on increasing the flexibility of the geometry type while still retaining parallelism. Also, compression may be a useful mechanism for decreasing the amount of data that needs to be sent across the network.

5.2 Rendering on the CM-5

Another way to alleviate the problem of having to send geometry data back to the workstation is to do the rendering on the CM-5. This is an area of active research. There are several ways to approach the problem, each with their own trade offs. The advantage of rendering directly on the CM-5 is that the data need not be transmitted across a network, only an image needs to be sent. The disadvantage is that the CM-5 probably will not be able to render as fast as dedicated graphics hardware, such as that found in graphics workstations, for small geometry databases. Where it should payoff, although, is when rendering tens of millions, or more, of primitives. With some of our simulations, we have generated polygonal databases over 200 megabytes in size.

Currently, we have Gouraud shaded polygonal renderers running in data parallel mode. At the same time, we have task parallel generalized polygonal renderers as well as volume renderers. These will be added to CM/AVS as CM/AVS modules using our specialized *geometry field* discussed previously.

5.3 UCD data in CM/AVS

As with geometrical data, the AVS UCD data type would also be useful on the CM-5. Again, the problem lies in how to deal affectively with mapping the data onto the supercomputer in a data parallel way. While most of our work has only needed the AVS field type to sufficiently support our simulation data, one case has already arose where the UCD data type would have been better.

6 Conclusion

CM/AVS has demonstrated that it is an affective environment for visualization on a massively parallel supercomputer. Together with the supercomputer and a graphics workstation, it makes best use of each machine's capabilities, whereas neither machine has adequate resources to do

Figure 2: Molecular Dynamics Visualization with CM/AVS

Figure 3: LANL Global Ocean Model Visualization with CM/AVS

the task alone. Although CM/AVS is still young, it provides the programmer with the necessary, general purpose, framework for prototyping and developing visualization methods and tools for massively parallel simulations.

7 Acknowledgements

The authors would like to thank James Salem, Matt Fitzgibbon, and Gary Oberbrunner from the visualization group at Thinking Machines Corporation for their support and allowing us to participate in the beta testing for CM/AVS.

We would also like to thank Dr. Robert Malone, Dr. Richard Smith, and Dr. Peter Lomdahl from Los Alamos National Laboratory for providing us with the data to visualize.

References

[1] Thinking Machines Corporation. The Connection Machine CM-5 technical summary, 1991.

[2] C. Upson, T. Faulhaber, D. Kamins, D. Schlegel, J. Vroom, R. Gurwitz, and A. van Dam. The Application Visualization System: A computational environment for scientific visualization. *IEEE Computer Graphics and Applications*, 9(4):30–42, July 1989.

[3] Advanced Visual Systems Inc., Waltham, MA. *AVS User's Guide*, release 4 edition, May 1992.

[4] Thinking Machines Corporation, Cambridge, MA. *CM/AVS User's Guide*, version 1.0 beta edition, October 1992.